Reflexão

António Menezes Leitão

24 de Março de 2008

Sistema Computacional

Definição

Um **Sistema Computacional** é um sistema que opera sobre um determinado domínio.

Definição

O domínio é representado pelas estruturas internas do sistema:

- Dados que representam as entidades e relações do domínio.
- Programa que descreve a manipulação dos dados.

Notas

- Um programa não é um sistema computacional.
- Um programa descreve (parte de) um sistema computacional.
- Um programa em execução é um sistema computacional.

Meta-Sistema Computacional

Definição

Um **Meta-Sistema Computacional** é um sistema computacional que tem como domínio um outro sistema computacional.

Definição

Um meta-sistema computacional manipula, como dados, uma representação do sistema computacional objecto.

Notas

- Um *debugger* é um meta-sistema computacional.
- Um *profiler* é um meta-sistema computacional.
- Um compilador (clássico) não é um meta-sistema computacional (o seu domínio é um programa e não um sistema computacional).

Reflexão

Definição

A **Reflexão** é a capacidade de uma entidade de se representar e operar do mesmo modo como representa e opera outros domínios.

Definição

Um **Sistema Computacional Reflexivo** é um meta-sistema computacional que se tem a si próprio como domínio.

Definição

Reflexão é a capacidade que um sistema computacional tem de manipular uma representação da sua estrutura e comportamento durante a sua própria execução.

Dois *graus* de Reflexão

Definição

A **Introspecção** é a capacidade de um sistema de *observar* a sua própria estrutura e comportamento.

Definição

A **Intercessão** é a capacidade de um sistema de *modificar* a sua própria estrutura e comportamento.

Exemplos

- ``Quantos parâmetros tem a função `foo`?'' é introspecção.
- ``Muda a classe desta instância para `Bar`!'' é intercessão.

Definição

A **reificação** é a criação de uma entidade que representa, no meta-sistema, uma entidade do sistema. A reificação é condição para a reflexão.

Exemplos

- ``Qual é a classe desta instância?'' implica a reificação de classes.
- ``Quais são os métodos desta classe?'' implica a reificação de métodos.
- ``Qual foi a cadeia de invocações que conduziu à invocação desta função?'' implica a reificação do *stack*.
- ``Quais são as variáveis livres desta função?'' implica a reificação do ambiente léxico.

Notas

- A **reificação** implica uma relação causal entre as entidades reificadas e a sua reificação. Qualquer modificação num implica a modificação do outro.
- Uma entidade reificada é uma entidade de **primeira classe** mas o contrário não é necessáriamente verdade.

Exemplos

- Em Scheme, Common Lisp e Emacs Lisp, uma função é uma entidade de primeira classe.
- Em Scheme, uma função não pode ser introspeccionada.
- Em Common Lisp, uma função pode ser (parcialmente) introspeccionada (`function-lambda-expression`, `disassemble`, `ed`).
- Em Emacs Lisp uma função (não compilada) pode ser introspeccionada.

Dois *graus* de reificação

Definição

A **Reificação estrutural** é capacidade do sistema de reificar a sua *estrutura*.

Definição

A **Reificação comportamental** (ou computacional) é capacidade do sistema de reificar a sua *execução*.

Exemplos

- ``Quais são as variáveis de instância desta classe?'' implica reificação estrutural.
- ``Quais são os *error handlers* que estão activos neste momento?'' implica reificação comportamental.

Programação: Linguagens *vs* Ambientes

Definição

Uma **Linguagem de Programação** é um meio através do qual se podem descrever rigorosamente processos computacionais.

Definição

Um **Programa** é a descrição de um processo computacional escrito numa linguagem de programação.

Definição

Um **Ambiente de Programação** é um sistema computacional destinado a auxiliar o desenvolvimento de programas.

Programação: Linguagens *vs* Ambientes

Notas

- Nem todas as linguagem de programação possuem suficientes mecanismos de reflexão.
- Mas a maioria dos ambientes de programação disponibilizam mecanismos alternativos para o mesmo efeito.
- Quando os programas podem ser executados autonomamente do ambiente, é importante distinguir os mecanismos que fazem parte da linguagem daqueles que fazem parte do ambiente.

Distinção Importante

- Os mecanismos que fazem parte do ambiente apenas podem ser usados durante o desenvolvimento.
- Os mecanismos que fazem parte da linguagem podem sempre ser usados.

Arquitectura Reflexiva

Definição

Um linguagem de programação possui uma **Arquitectura Reflexiva** quando trata a reflexão como um conceito fundamental e providencia mecanismos explícitos para o programador a utilizar.

Implicações

- O processador da linguagem disponibiliza dados que representam o próprio sistema.
- Os programas podem descrever computações reflexivas sobre esses dados.
- Existe uma relação causal entre esses dados e o estado e comportamento do sistema.
- Modificações feitas aos dados que representam o sistema são modificações ao sistema.

Arquitectura Reflexiva

Notas

- A reificação comportamental é muito mais difícil de implementar eficientemente que a reificação estrutural.
- A intercessão sobre a reificação comportamental complica substancialmente a compilação.

Questões

- Como representar a semântica de uma linguagem de modo a que programas escritos nessa linguagem a possam modificar?
- Como reificar preservando a eficiência?
- Como compilar programas cuja semântica pode ser modificada durante a sua execução?

Linguagens com Arquitectura Reflexiva

Lisp

- Mote: ``*Igualdade entre dados e programas*''.
- Permite inspeccionar os programas como se fossem dados.
- Permite construir programas a partir de dados.

Smalltalk

- Mote: ``*Tudo são objectos*''.
- Tudo construído do mesmo modo (classes e métodos), incluíndo compilador, máquina virtual, IDE, etc.
- Permite introspecção e intercessão generalizadas sobre classes, instâncias, compilador, *stack*, etc.

Emacs Lisp

Exemplo

```
> (defun foo (x)                                   ;definição da função
    (+ x 3))
foo
> (foo 4)                                          ;invocação da função
7
> (symbol-function 'foo)                           ;a própria função
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))                    ;os seus parâmetros
(x)
> (caaddr (symbol-function 'foo))                  ;a função invocada
+
> (setf (caaddr (symbol-function 'foo)) '-)        ;alteremo-la
-
> (foo 4)                                          ;invocação da função
1
> (symbol-function 'foo)                           ;a 'nova' função
(lambda (x)
  (- x 3))
```

Emacs Lisp

Definição (*Trace* / Rastreio)

- A cada invocação de uma dada função, é escrito que foi invocada, quais os argumentos e qual o resultado.
- É uma forma de introspecção comportamental.
- Para simplificar, vamos omitir o resultado da invocação.

Trace

```
> (defun fact (n)
    (if (= n 0)
      1
      (* n (fact (- n 1)))))
fact
> (fact 4)
24
> (trace 'fact)
fact
> (fact 4)
(fact 4)->(fact 3)->(fact 2)->(fact 1)->(fact 0)->24
```

Emacs Lisp

Implementação

- Detecção da invocação e rastreio realizado pelo interpretador.
- Injecção de código de rastreio na função.
- Redefinição da função para incluir código de rastreio.

Trace por Redefinição

```
(defun trace (name)
  (setf (symbol-function name)
        (traced-lambda name
                       (symbol-function name)))
  name)
```

Nota (defun)

(defun *name* (arg_0 arg_1 … arg_n) *body*)

(setf (symbol-function *name*) (lambda (arg_0 arg_1 … arg_n) *body*))

Emacs Lisp

Original

```
(lambda (n)
  (if (= n 0)
    1
    (* n (fact (- n 1)))))
```

Com Rastreio

```
(lambda (n)
  (princ (list 'fact n))
  (princ '->)
  (if (= n 0)
    1
    (* n (fact (- n 1)))))
```

Inclusão do Rastreio

```
(defun traced-lambda (name lambda-form)
  (cons 'lambda
        (cons (cadr lambda-form)
              (cons (list 'princ
                          (cons 'list
                                (cons (list 'quote name)
                                      (cadr lambda-form))))
                    (cons '(princ '->)
                          (cddr lambda-form))))))
```

Backquote (1978)

Definição

- Facilita a escrita de meta-programas:
- ` significa ``não avaliar'' a expressão seguinte (tal como o *quote* ') excepto quando surgir uma vírgula,
- , significa avaliar a expressão seguinte e *inserir* o valor resultante na expressão envolvente,
- ,@ significa avaliar a expressão seguinte e *espalhar* a lista resultante na expressão envolvente.

Exemplo

```
> '(5 (list (+ 1 3) 3) 2 1)
(5 (list (+ 1 3) 3) 2 1)
> `(5 ,(list (+ 1 3) 3) 2 1)
(5 (4 3) 2 1)
> `(5 ,@(list (+ 1 3) 3) 2 1)
(5 4 3 2 1)
```

Backquote

De

```
(lambda (n)
  (if …))
```

Para

```
(lambda (n)
  (princ (list 'fact n))
  (princ '->)
  (if …))
```

Backquote

De

```
(lambda (n)
  (if ...))
```

Para

```
(lambda (n)
  (princ (list 'fact n))
  (princ '->)
  (if ...))
```

Sem *backquote*

```
(cons 'lambda
      (cons (cadr lambda-form)
            (cons (list 'princ
                        (cons 'list
                              (cons (list 'quote name)
                                    (cadr lambda-form))))
                  (cons '(princ '->)
                        (cddr lambda-form)))))
```

Backquote

De

```
(lambda (n)
  (if …))
```

Para

```
(lambda (n)
  (princ (list 'fact n))
  (princ '->)
  (if …))
```

Com *backquote*

```
`(lambda ,(cadr lambda-form)
   (princ (list ',name ,@(cadr lambda-form)))
   (princ '->)
   ,@(cddr lambda-form))
```

Duplo Backquote

Definição

- Na presença de dois *backquotes* encadeado, a vírgula mais interna está associada ao *backquote* mais externo.
- `,` significa avaliar a expressão seguinte quando o *backquote* interior é avaliado e *inserir* o valor resultante na expressão envolvente,
- `,,` significa avaliar a expressão seguinte duas vezes e *inserir* o valor resultante na expressão envolvente,
- `,',` significa avaliar a expressão seguinte quando o *backquote* exterior é avaliado e *inserir* o valor resultante na expressão envolvente,
- `,@,` significa avaliar a expressão seguinte duas vezes e *espalhar* o valor resultante na expressão envolvente,

Auto Modificação

Exemplo (Crescimento Exponencial)

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
................................832040
> (fib 40)
.................................................................
```

Auto Modificação

Memoization

```
(defun memoize (lambda-form)
  (setcar (cddr lambda-form)
          `(let ((result ,(caddr lambda-form)))
             (setcar (cddr ',lambda-form)
                     `(if (eql ,',(caadr lambda-form)
                               ',,(caadr lambda-form))
                        ',result
                        ,(caddr ',lambda-form)))
             result)))
```

Exemplo

```
> (memoize (symbol-function 'fib))

> (fib 10)
55
> (fib 20)
6765
> (fib 40)
102334155
```

Evolução da função fib

Após definição

```
(lambda (n)
  (if (< n 2)
    n
    (+ (fib (- n 1))
       (fib (- n 2)))))
```

Evolução da função `fib`

Após *memoization*

```
(lambda (n)
  (let ((result
          (if (< n 2)
            n
            (+ (fib (- n 1)) (fib (- n 2))))))
    (setcar (cddr '(lambda (n) ...))
            (list 'if
                  (list 'eql 'n (list 'quote n))
                  (list 'quote result)
                  (caddr '(lambda (n) ...))))
    result))
```

Evolução da função fib

Após (fib 0)

```
(lambda (n)
  (if (eql n '0)
    '0
    (let ((result
            (if (< n 2)
              n
              (+ (fib (- n 1)) (fib (- n 2))))))
      (setcar (cddr '(lambda (n) ...))
              (list 'if
                    (list 'eql 'n (list 'quote n))
                    (list 'quote result)
                    (caddr '(lambda (n) ...))))
      result)))
```

Evolução da função fib

Após (fib 1)

```
(lambda (n)
  (if (eql n '1)
    '1
    (if (eql n '0)
      '0
      (let ((result
              (if (< n 2)
                n
                (+ (fib (- n 1)) (fib (- n 2))))))
        (setcar (cddr '(lambda (n) ...))
                (list 'if
                      (list 'eql 'n (list 'quote n))
                      (list 'quote result)
                      (caddr '(lambda (n) ...))))
        result))))
```

Evolução da função fib

Após (fib 40)

```
(lambda (n)
  (if (eql n '40)
    '102334155
    (if (eql n '39)
      '63245986
      (if (eql n '38)
        '39088169
          …
                (if (eql n '3)
                  '2
                  (if (eql n '2)
                    '1
                    (if (eql n '1)
                      '1
                      (if (eql n '0)
                        '0
                        (let ((result
                                (if (< n 2)
                                  n
                                  (+ (fib (- n 1)) (fib (- n 2))))))
                          (setcar (cddr '(lambda (n) …))
                                  (list 'if
                                        (list 'eql 'n (list 'quote n))
                                        (list 'quote result)
                                        (caddr '(lambda (n) …))))
                          result))))))))))))
```

Linguagens com Arquitectura Reflexiva

- A intercessão ilimitada coloca imenso poder nas mãos do programador:
  - É possível modificar dinamicamente os programas.
  - É possível escrever programas que se auto-modificam.
- Imenso poder implica imensa responsabilidade:
  - Qual a semântica de um programa que se pode modificar durante a sua execução?
  - Como depurar um programa cuja forma foi sendo progressivamente (auto) modificada?
  - Como compilar um programa que não possui uma forma estável?
- Lisp evoluiu no sentido de regular as capacidades de intercessão de modo a permitir compilação eficiente.
- Mas o poder continua lá!

Java

Características

- Sintaticamente descendente de C++.
- Semanticamente descendente de Smalltalk.
- Enorme divulgação.
- Começou por não ter quaisquer capacidades reflexivas que têm vindo a ser adicionadas ao longo das sucessivas versões.
- Actualmente, permite introspecção estrutural e formas muito limitadas de intercessão comportamental.
- A reflexão opera sobre os elementos da linguagem: classes, *fields*, construtores, métodos, etc.

Reflexão em Java

Constructores e métodos não privados de uma classe

```
import java.lang.reflect.*;

public class PrintClass {

    public static void main(String[] args)
      throws ClassNotFoundException {
        if (args.length != 1) {
            System.err.println("Usage: java PrintClass <class>");
            System.exit(1);
        } else {
            dumpClass(Class.forName(args[0]));
        }
    }

    static void dumpClass(Class c) {
        ...
    }
}
```

Reflexão em Java

Constructores e métodos não privados de uma classe

```
static void dumpClass(Class c) {
    System.out.println(c + " {");

    for (Constructor con : c.getConstructors()) {
        System.out.println("    " + con);
    }

    for (Method m : c.getDeclaredMethods()) {
        if (! Modifier.isPrivate(m.getModifiers())) {
            System.out.println("    " + m);
        }
    }

    System.out.println("}");
}
```

Reflexão em Java

Constructores e métodos não privados de uma classe

```
$ java PrintClass java.lang.Object
class java.lang.Object {
    public java.lang.Object()
    public native int java.lang.Object.hashCode()
    public final native java.lang.Class java.lang.Object.getClass()
    …
    public boolean java.lang.Object.equals(java.lang.Object)
    public final native void java.lang.Object.notify()
    public final native void java.lang.Object.notifyAll()
    public java.lang.String java.lang.Object.toString()
}
```

Reflexão em Java

- Em Java, os tipos dividem-se em:
  - Tipos primitivos: `boolean`, `byte`, `short`, `int`, `long`, `char`, `float` e `double`.
  - Tipos referência: `java.lang.String`, `java.io.Serializable`, `java.lang.Integer` e todos os outros.
- Para cada tipo (primitivo ou referência), existe uma instância única da classe `java.lang.Class` que representa esse tipo.
- A classe `java.lang.Class` possui vários métodos que permitem:
  - obter informações (métodos, variáveis, etc),
  - criar instâncias,
  - alterar variáveis e invocar métodos.

Reflexão em Java

Para se obter uma instância de `java.lang.Class`

- A partir de um objecto *foo*:
  `foo.getClass()`
- A partir de um tipo *Bar*:
  `Bar.class`
- A partir do nome de um tipo `"foo.bar.Baz"` (se não encontrar, assinala a *Checked exception* `ClassNotFoundException`):
  `Class.forName("foo.bar.Baz")`

Exemplo

```
"I am a string".getClass()

String.class

Class.forName("java.lang.String")
```

Reflexão em Java

Métodos importantes na classe `Class`

- `boolean isPrimitive()`
  Determines if the type represented by the receiver is a primitive type.
- `boolean isInterface()`
  Determines if the type represented by the receiver represents an interface type.
- `boolean isArray()`
  Determines if the type represented by the receiver is an array class.
- `Class getComponentType()`
  Returns the Class representing the component type of the array class represented by the receiver.
- `String getName()`
  Returns the name of the entity (class, interface, array class, primitive type, or void) represented by the receiver, as a String.

Reflexão em Java

Métodos importantes na classe `Class`

- `Package getPackage()`
  Gets the package of type represented by the receiver.
- `int getModifiers()`
  Returns the Java language modifiers for the type represented by the receiver, encoded in an integer.
- `Class getSuperclass()`
  Returns the Class representing the superclass of the class represented by the receiver.
- `Class[] getInterfaces()`
  Determines the Classes representing the interfaces implemented by the class or interface represented by the receiver.
- `Class getDeclaringClass()`
  If the class or interface represented by the receiver is a member of another class, returns the Class object representing that class.

Reflexão em Java

Métodos importantes na classe `Class`

- `Class[] getClasses()`
  Returns an array containing `Classes` representing all the public classes and interfaces members of the class represented by the receiver.
- `Field[] getFields()`
  Returns an array containing `Fields` representing all the accessible public fields of the class or interface represented by the receiver.
- `Constructor[] getConstructors()`
  Returns an array containing `Constructors` representing all the public constructors of the class represented by the receiver.
- `Method[] getMethods()`
  Returns an array containing `Methods` representing all the public member methods of the class or interface represented by the receiver, including those declared by the class or interface and those inherited from superclasses and superinterfaces.

Reflexão em Java

Métodos importantes na classe `Class`

- `Class[] getDeclaredClasses()`
  Returns an array of `Classes` representing all the classes and interfaces declared as members of the class represented by the receiver.
- `Field[] getDeclaredFields()`
  Returns an array of `Fields` reflecting all the fields declared by the class or interface represented by the receiver.
- `Constructor[] getDeclaredConstructors()`
  Returns an array of `Constructors` representing all the constructors declared by the class represented by the receiver.
- `Method[] getDeclaredMethods()`
  Returns an array of `Methods` reflecting all the methods declared by the class or interface represented by the receiver.

Reflexão em Java

Métodos importantes na classe `Class`

- `Field getField(String name)`
  Returns a `Field` that represents the specified public member field of the class or interface represented by the receiver.
- `Field getDeclaredField(String name)`
  Returns a `Field` representing the specified declared field of the class or interface represented by the receiver.
- `Constructor getConstructor(Class[] types)`
  Returns a `Constructor` that represents the specified public constructor of the class represented by the receiver.
- `Constructor getDeclaredConstructor(Class[] types)`
  Returns a `Constructor` that represents the specified constructor of the class represented by the receiver.

Reflexão em Java

Métodos importantes na classe `Class`

- `Method getMethod(String name, Class[] types)`
  Returns a `Method` that represents the specified public member method of the class or interface represented by the receiver.
- `Method getDeclaredMethod(String name, Class[] types)`
  Returns a `Method` that represents the specified declared method of the class or interface represented by the receiver.
- `boolean isAssignableFrom(Class cls)`
  Determines if the class or interface represented by the receiver is either the same as, or is a superclass or superinterface of, the class or interface represented by the specified `Class` parameter.
- `Object newInstance()`
  Creates a new instance of the class represented by the receiver.
- `boolean isInstance(Object obj)`
  Determines if the specified Object is assignment-compatible with the type represented by the receiver.

Reflexão em Java

Atenção aos *packages*

- `java.lang.Object`
- `java.lang.Class`
- `java.lang.Package`
- `java.reflect.Field`
- `java.reflect.Constructor`
- `java.reflect.Method`
- `java.reflect.Modifier`

Reflexão em Java

Problema: produzir uma interface a partir de uma classe.

Carros

```
public class Car {

    public void move() { ... }

    public void stop() { ... }
}
```

Bicicletas

```
public class Bicycle {

    public void move() { ... }

    public void stop() { ... }
}
```

```
$ java GenerateInterfaceFromClass MovableThing Car
```

Interface

```
public interface MovableThing  {
    public abstract void stop ();
    public abstract void move ();
}
```

Reflexão em Java

Produzir uma interface a partir de uma classe

```
import java.lang.reflect.*;
import java.io.*;

public class GenerateInterfaceFromClass {
    public static void main(String[] args) throws IOException {
        if (args.length == 2) {
            generateInterfaceFromClass(args[0], getClass(args[1]));
        } else {
            println("Usage: java GenerateInterfaceFromClass <interface> <class>");
            System.exit(1);
        }
    }

    …

    protected static void println(String text) {
        System.out.println(text);
    }

    protected static void printsp(String text) {
        System.out.print(text);
        System.out.print(" ");
    }

    protected static void print(String text) {
        System.out.print(text);
    }
}
```

Reflexão em Java

Produzir uma interface a partir de uma classe

```
static Class getClass(String className) {
    try {
        return (Class.forName(className,
                              false,
                              GenerateInterfaceFromClass.class.getClassLoader()));
        // the false means don't initialize the class (don't
        // initialize static fields, don't execute static
        // initializers)
    } catch (ClassNotFoundException cnfe) {
        System.err.println("Class '" + className + "' not found");
        System.exit(1);
        return null;
    }
}

static void generateInterfaceFromClass(String interfaceName, Class fromClass) {
    printsp("public interface");
    printsp(interfaceName);
    printSuperInterfaces(fromClass.getInterfaces());
    println(" {");
    printMethods(fromClass.getDeclaredMethods());
    println("}");
}
```

Reflexão em Java

Produzir uma interface a partir de uma classe

```
static void printSuperInterfaces(Class[] interfaces) {
    if (interfaces.length > 0) {
        printsp("extends");
        for(int i = 0; i < interfaces.length; i++) {
            if (i > 0) {
                printsp(",");
            }
            print(toTypeName(interfaces[i]));
        }
    }
}

static void printMethods(Method[] methods) {
    for(Method method : methods) {
        if (! method.isBridge()) {
            int mods = method.getModifiers();
            if (Modifier.isPublic(mods)) {
                printsp("    public abstract");
                printsp(toTypeName(method.getReturnType()));
                printsp(method.getName());
                printParamList(method.getParameterTypes());
                printThrows(method.getExceptionTypes());
                println(";");
            }
        }
    }
}
```

Reflexão em Java

Produzir uma interface a partir de uma classe

```
static void printParamList(Class[] argTypes) {
    print("(");
    for(int i = 0; i < argTypes.length; i++) {
        if (i > 0) {
        printsp(",");
        }
        print(toTypeName(argTypes[i]) + " arg" + i);
    }
    print(")");
}

static void printThrows(Class[] excepTypes) {
    if (excepTypes.length > 0) {
        printsp(" throws");
        for(int i = 0; i < excepTypes.length; i++) {
            if (i > 0) {
                printsp(",");
            }
            print(toTypeName(excepTypes[i]));
        }
    }
}

static String toTypeName(Class classObj) {
    if (classObj.isArray()) {
        return toTypeName(classObj.getComponentType()) + "[]";
    } else {
        return classObj.getName();
    }
}
```

Reflexão em Java

Exemplo

```
class Shape {
}

class Line extends Shape {
}

class Circle extends Shape {
}

class Device {
    public void draw(Shape s) {
        System.err.println("draw what where?");
    }
    public void draw(Line l) {
        System.err.println("draw a line where?");
    }
    public void draw(Circle c)  {
        System.err.println("draw a circle where?");
    }
}
```

Reflexão em Java

Exemplo

```
class Screen extends Device {
    public void draw(Shape s) {
        System.err.println("draw what on screen?");
    }
    public void draw(Line l) {
        System.err.println("drawing a line on screen!");
    }
    public void draw(Circle c) {
        System.err.println("drawing a circle on screen!");
    }
}

class Printer extends Device {
    public void draw(Shape s) {
        System.err.println("draw what on printer?");
    }
    public void draw(Line l) {
        System.err.println("drawing a line on printer!");
    }
    public void draw(Circle c) {
        System.err.println("drawing a circle on printer!");
    }
}
```

Reflexão em Java

Qual é o *output?*

```
Shape[] shapes = new Shape[] { new Line(), new Circle() };

Device[] devices = new Device[] { new Screen(), new Printer() };

for (Device device : devices) {
    for (Shape shape : shapes) {
        device.draw(shape);
    }
}
```

Reflexão em Java

Qual é o *output?*

```
Shape[] shapes = new Shape[] { new Line(), new Circle() };

Device[] devices = new Device[] { new Screen(), new Printer() };

for (Device device : devices) {
    for (Shape shape : shapes) {
        device.draw(shape);
    }
}
```

Output

```
draw what on screen?
draw what on screen?
draw what on printer?
draw what on printer?
```

Reflexão em Java

Qual é o *output?*

```
Shape[] shapes = new Shape[] { new Line(), new Circle() };

Device[] devices = new Device[] { new Screen(), new Printer() };

for (Device device : devices) {
    for (Shape shape : shapes) {
        device.draw(shape);
    }
}
```

Output

```
draw what on screen?
draw what on screen?
draw what on printer?
draw what on printer?
```

Java emprega despacho dinâmico para o receptor e estático para os argumentos.

Reflexão em Java

Solução: *TypeCasts*

```
class Device {

    public void draw(Shape s) {
        if (s instanceof Line) {
            draw((Line)s);
        } else if (s instanceof Circle) {
            draw((Circle)s);
        } else {
            System.err.println("draw what where?");
        }
    }

    public void draw(Line l) {
        System.err.println("draw a line where?");
    }

    public void draw(Circle c)  {
        System.err.println("draw a circle where?");
    }
}
```

Reflexão em Java

Solução: *TypeCasts*

```
class Screen extends Device {

    public void draw(Line l) {
        System.err.println("drawing a line on screen!");
    }

    public void draw(Circle c) {
        System.err.println("drawing a circle on screen!");
    }
}

class Printer extends Device {

    public void draw(Line l) {
        System.err.println("drawing a line on printer!");
    }

    public void draw(Circle c) {
        System.err.println("drawing a circle on printer!");
    }
}
```

Reflexão em Java

Problemas

- É mais eficiente desenhar instâncias de `Line` (um teste) do que instâncias de `Circle` (dois testes).
- Quando as subclasses de `Shape` formam uma hierarquia é preciso pensar cuidadosamente na ordem dos testes no método `draw`.
- Sempre que se acrescenta uma nova subclasse de `Shape` é preciso modificar o método `draw` (e repensar a ordem dos testes).

Reflexão em Java

Problemas

- É mais eficiente desenhar instâncias de `Line` (um teste) do que instâncias de `Circle` (dois testes).
- Quando as subclasses de `Shape` formam uma hierarquia é preciso pensar cuidadosamente na ordem dos testes no método `draw`.
- Sempre que se acrescenta uma nova subclasse de `Shape` é preciso modificar o método `draw` (e repensar a ordem dos testes).

Solução

Despacho duplo.

Reflexão em Java

Solução: Despacho duplo

```
abstract class Device {

    public abstract void draw(Shape s);
}

class Screen extends Device {

    public void draw(Shape s) {
        s.drawOnScreen(this);
    }
}

class Printer extends Device {

    public void draw(Shape s) {
        s.drawOnPrinter(this);
    }
}
```

Reflexão em Java

Solução: Despacho duplo

```
abstract class Shape {
    public abstract void drawOnScreen(Screen s);
    public abstract void drawOnPrinter(Printer p);
}

class Line extends Shape {
    public void drawOnScreen(Screen s) {
        System.err.println("drawing a line on screen!");
    }
    public void drawOnPrinter(Printer p) {
        System.err.println("drawing a line on printer!");
    }
}

class Circle extends Shape {
    public void drawOnScreen(Screen s) {
        System.err.println("drawing a circle on screen!");
    }
    public void drawOnPrinter(Printer p) {
        System.err.println("drawing a circle on printer!");
    }
}
```

Reflexão em Java

Solução: Despacho duplo + *Overloading*

```
abstract class Shape {
    public abstract void draw(Screen s);
    public abstract void draw(Printer p);
}

class Line extends Shape {
    public void draw(Screen s) {
        System.err.println("drawing a line on screen!");
    }
    public void draw(Printer p) {
        System.err.println("drawing a line on printer!");
    }
}

class Circle extends Shape {
    public void draw(Screen s) {
        System.err.println("drawing a circle on screen!");
    }
    public void draw(Printer p) {
        System.err.println("drawing a circle on printer!");
    }
}
```

Reflexão em Java

Solução: Despacho duplo

```
abstract class Device {

    public abstract void draw(Shape s);
}

class Screen extends Device {

    public void draw(Shape s) {
        s.drawOnScreen(this);
    }
}

class Printer extends Device {

    public void draw(Shape s) {
        s.drawOnPrinter(this);
    }
}
```

Reflexão em Java

Solução: Despacho duplo + *Overloading*

```
abstract class Device {

    public abstract void draw(Shape s);
}

class Screen extends Device {

    public void draw(Shape s) {
        s.draw(this);
    }
}

class Printer extends Device {

    public void draw(Shape s) {
        s.draw(this);
    }
}
```

Reflexão em Java

Problemas

- Implica reestruturação do programa
- É fácil criar um novo tipo de `Shape` mas um novo tipo de `Device` implica acrescentar um método a todos os tipos de `Shape`.
- Cada subclasse the `Device` precisa de ter uma cópia do método `draw`.
- Ordem de despacho fixa: primeiro, por tipo de `Device`, depois por tipo de `Shape`.
- A generalização para despacho triplo, quádruplo, etc, provoca uma explosão combinatória de métodos.

Reflexão em Java

Problemas

- Implica reestruturação do programa
- É fácil criar um novo tipo de `Shape` mas um novo tipo de `Device` implica acrescentar um método a todos os tipos de `Shape`.
- Cada subclasse the `Device` precisa de ter uma cópia do método `draw`.
- Ordem de despacho fixa: primeiro, por tipo de `Device`, depois por tipo de `Shape`.
- A generalização para despacho triplo, quádruplo, etc, provoca uma explosão combinatória de métodos.

Solução

Controlar o mecanismo de invocação.

Reflexão em Java

Invocação Dinâmica

```
class Device {

    public void draw(Shape s) {
        invoke(this, "draw", s);
    }
    public static Object invoke(Object receiver,
                                String name,
                                Object arg) {
        try {
            Method method = findBestMethod(receiver.getClass(),
                                           name,
                                           arg.getClass());
            return method.invoke(receiver,
                                 new Object[] { arg });
        } catch (NoSuchMethodException e) {
            throw new RuntimeException(e);
        } catch (IllegalAccessException e) {
            throw new RuntimeException(e);
        } catch (InvocationTargetException e) {
            throw new RuntimeException(e);
        }
    }
    ...
}
```

Reflexão em Java

Invocação Dinâmica

```
class Device {

    …

    public static Method findBestMethod(Class type,
                                        String name,
                                        Class argType)
        throws NoSuchMethodException {
        try {
            return type.getMethod(name, new Class[] { argType });
        } catch (NoSuchMethodException e) {
            if (argType == Object.class) {
                throw new NoSuchMethodException(name);
            } else {
                return findBestMethod(type,
                                      name,
                                      argType.getSuperclass());
            }
        }
    }
}
```

Reflexão em Java

Problemas

- `getMethod` apenas acede a métodos **públicos**.
- `findBestMethod` apenas acede a métodos públicos com um único parâmetro.
- Estamos a ``trepar'' na hieraquia de classes mas não estamos ``trepar'' na hierarquia de interfaces.
- Não estamos a lidar com *boxing*/*unboxing*.
- Não estamos a lidar com métodos de aridade variável.

Reflexão em Java

Problemas

- `getMethod` apenas acede a métodos **públicos**.
- `findBestMethod` apenas acede a métodos públicos com um único parâmetro.
- Estamos a ``trepar'' na hieraquia de classes mas não estamos ``trepar'' na hierarquia de interfaces.
- Não estamos a lidar com *boxing/unboxing*.
- Não estamos a lidar com métodos de aridade variável.

Solução

Mais trabalho!

Reflexão em Java

Invocação de Métodos em Java

De acordo com a *Java Language Specification*:

- primeira edição: 5186 palavras

Resumo

Reflexão em Java

Invocação de Métodos em Java

De acordo com a *Java Language Specification*:

- primeira edição: 5186 palavras
- segunda edição: 5473 palavras

Resumo

Reflexão em Java

Invocação de Métodos em Java

De acordo com a *Java Language Specification*:

- primeira edição: 5186 palavras
- segunda edição: 5473 palavras
- terceira edição: 13284 palavras

Resumo

Reflexão em Java

Invocação de Métodos em Java

De acordo com a *Java Language Specification*:

- primeira edição: 5186 palavras
- segunda edição: 5473 palavras
- terceira edição: 13284 palavras

Resumo

1. Determine Class or Interface to Search.

Reflexão em Java

Invocação de Métodos em Java

De acordo com a *Java Language Specification*:

- primeira edição: 5186 palavras
- segunda edição: 5473 palavras
- terceira edição: 13284 palavras

Resumo

1. Determine Class or Interface to Search.
2. Determine Method Signature.

Reflexão em Java

Invocação de Métodos em Java

De acordo com a *Java Language Specification*:

- primeira edição: 5186 palavras
- segunda edição: 5473 palavras
- terceira edição: 13284 palavras

Resumo

1. Determine Class or Interface to Search.
2. Determine Method Signature.
3. Identify Matching Arity Methods Applicable by Subtyping.

Reflexão em Java

Invocação de Métodos em Java

De acordo com a *Java Language Specification*:

- primeira edição: 5186 palavras
- segunda edição: 5473 palavras
- terceira edição: 13284 palavras

Resumo

1. Determine Class or Interface to Search.
2. Determine Method Signature.
3. Identify Matching Arity Methods Applicable by Subtyping.
4. Identify Matching Arity Methods Applicable by Method Invocation Conversion.

Reflexão em Java

Invocação de Métodos em Java

De acordo com a *Java Language Specification*:

- primeira edição: 5186 palavras
- segunda edição: 5473 palavras
- terceira edição: 13284 palavras

Resumo

1. Determine Class or Interface to Search.
2. Determine Method Signature.
3. Identify Matching Arity Methods Applicable by Subtyping.
4. Identify Matching Arity Methods Applicable by Method Invocation Conversion.
5. Identify Applicable Variable Arity Methods.

Reflexão em Java

Invocação Dinâmica

```
public static Object invoke(Object receiver,
                            String name,
                            Object... args) {
    try {
        Method method = findBestMethodIndex(receiver.getClass(),
                                            name,
                                            getTypes(args),
                                            args.length - 1);
        return method.invoke(receiver, args);
    } catch (IllegalAccessException e) {
        throw new RuntimeException(e);
    } catch (InvocationTargetException e) {
        throw new RuntimeException(e);
    }
}

static Class[] getTypes(Object[] args) {
    Class types[] = new Class[args.length];

    for (int i = 0; i < args.length; i++) {
        types[i] = args[i].getClass();
    }
    return types;
}
```

Reflexão em Java

Invocação Dinâmica

```
static Method findBestMethodIndex(Class receiverType, String name,
                                  Class[] argTypes, int index) {
    if (index < 0) {
        return null;
    } else {
        Class originalType=argTypes[index];
        Method method = null;
        while (method == null) {
            try {
                method=receiverType.getMethod(name, argTypes);
            } catch(NoSuchMethodException e) { }
            if (method != null) {
                break;
            } else {
                argTypes[index] = argTypes[index].getSuperclass();
                if (argTypes[index]==null) {
                    break;
                }
            }
            method=findBestMethodIndex(receiverType, name,
                                       argTypes, index-1);
        }
        argTypes[index] = originalType;
        return method;
    }
}
```

Reflexão em Java

Exemplo

```
class Calculator {

    public void print(Object o) {
        System.out.println("" + o);
    }

    public void join(Object a, Object b) {
        Vector v = new Vector();
        v.add(a);
        v.add(b);
        print(v);
    }

    public void join(String a, Object b) {
        print(a + b);
    }

    public void join(Integer a, Integer b) {
        print(a + b);
    }
}
```

Reflexão em Java

Exemplo

```
Calculator calc = new Calculator();
Object[] objs1 = new Object[] { "Hello", 1, 'A' };
Object[] objs2 = new Object[] { "World", 2, 'B' };
for (Object o1 : objs1) {
    for (Object o2 : objs2) {
        invoke(calc, "join", o1, o2);
    }
}
```

Exemplo

```
HelloWorld
Hello2
HelloB
[1, World]
3
[1, B]
[A, World]
[A, 2]
[A, B]
```

Modificação de Programas

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}
```

Exemplo (Crescimento Exponencial)

```
$ java Fib 10
..55
$ java Fib 20
........6765
$ java Fib 30
................................832040
$ java Fib 40
................................................................
```

Modificação de Programas com Javassist

Memoization

```
import javassist.*;
import java.io.*;

public class Memoize {

  public static void main(String[] args)
    throws NotFoundException, CannotCompileException, IOException {
    if (args.length != 2) {
        System.err.println("Usage: java Memoize <class> <method>");
        System.exit(1);
    } else {
        ClassPool pool = ClassPool.getDefault();
        CtClass ctClass = pool.get(args[0]);
        memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
        ctClass.writeFile();
    }
  }

  static void memoize(CtClass ctClass, CtMethod ctMethod) {
     ...
  }
}
```

Modificação de Programas com Javassist

Memoization

```
static void memoize(CtClass ctClass, CtMethod ctMethod)
    throws NotFoundException, CannotCompileException {
    CtField ctField =
        CtField.make("static java.util.Hashtable cachedResults = " +
                     "    new java.util.Hashtable();",
                     ctClass);
    ctClass.addField(ctField);
    String name = ctMethod.getName();
    ctMethod.setName(name + "$original");
    ctMethod = CtNewMethod.copy(ctMethod, name, ctClass, null);
    ctMethod.setBody("{" +
                     "  Object result = cachedResults.get($1);" +
                     "  if (result == null) {" +
                     "    result = " + name + "$original($$);" +
                     "    cachedResults.put($1, result);" +
                     "  }" +
                     "  return ($r)result;" +
                     "}");
    ctClass.addMethod(ctMethod);
}
```

Modificação de Programas com Javassist

Exemplo

Resultados

```
$ time java Fib 40
102334155

real    0m13.784s
user    0m12.521s
sys     0m0.056s

$ java -classpath ".:javassist.jar" Memoize Fib fib

$ time java Fib 40
102334155

real    0m0.093s
user    0m0.036s
sys     0m0.012s
```

Evolução da função fib

Pré *Memoization*

```
public class Fib {

    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
}
```

Evolução da função `fib`

Pós *Memoization*

```
public class Fib {

    public static Long fib$original (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }

    static Hashtable cachedResults = new Hashtable();

    public static Long fib (Long n) {
        Object result = cachedResults.get(n);
        if (result == null) {
            result = fib$original(n);
            cachedResults.put(n, result);
        }
        return (Long)result;
    }
}
```

Meta-Variáveis em Javassist

Definição

- O código a inserir é uma *template* de código Java (numa `String`) que pode conter *meta-variáveis*.
- A *template* pode representar um *statement* (quando termina em `;`) ou um bloco (quando contida em `{}`).
- O Javassist compila a *template*, atribuindo um significado especial às meta-variáveis:
- `$0` é o receptor (inexistente em métodos estáticos).
- `$1`,`$2`,`$3`, etc, é cada um dos parâmetros do método (os nomes não são acessíveis). Pode-se ler e atribuir.
- `$$` é todos os parâmetros, i.e., `$1,$2,...`.
- `$r` é o tipo de retorno do método e pode ser usado em *casts*.
- `$w` é o tipo *wrapper* e pode ser usado em *casts* de tipos primitivos.

Javassist

Problemas

- *Templates* de código baseados na concatenação de `Strings` são frágeis.
- O compilador do Javassist é muito frágil e apenas lida com Java 1.0 e fragmentos das versões superiores.
- Pode-se violar a semântica do Java (tipo de retorno incorrecto, omissão de *type casts*, despacho errado, etc).
- O verificador de *byte-code* da JVM ainda pode conseguir apanhar alguns dos erros (em *run-time*) mas não há garantia.
- Não é prático ter de "recompilar"manualmente os *class files*

Javassist

Problemas

- *Templates* de código baseados na concatenação de `Strings` são frágeis.
- O compilador do Javassist é muito frágil e apenas lida com Java 1.0 e fragmentos das versões superiores.
- Pode-se violar a semântica do Java (tipo de retorno incorrecto, omissão de *type casts*, despacho errado, etc).
- O verificador de *byte-code* da JVM ainda pode conseguir apanhar alguns dos erros (em *run-time*) mas não há garantia.
- Não é prático ter de "recompilar"manualmente os *class files*

Solução (para o último problema)

Modificação de classes em *load time*.

Modificação de Programas em *Load Time*

Transfere controle para o main da classe modificada

```
import javassist.*;
import java.io.*;
import java.lang.reflect.*;

public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
            Class rtClass = ctClass.toClass();
            Method main =
                rtClass.getMethod("main",
                                  new Class[] { args.getClass() });
            String[] restArgs = new String[args.length - 2];
            System.arraycopy(args, 2, restArgs, 0, restArgs.length);
            main.invoke(null, new Object[] { restArgs });
        }
    }
}
```

Modificação de Programas em *Load Time*

Exemplo

Resultados

```
$ time java Fib 40
102334155

real    0m0.093s
user    0m0.036s
sys     0m0.012s
$ javac Fib.java
$ time java Fib 40
102334155

real    0m13.501s
user    0m12.509s
sys     0m0.032s
$ time java -classpath ".:javassist.jar" MemoizeAndRun Fib fib 40
102334155

real    0m0.381s
user    0m0.268s
sys     0m0.032s
```

Modificação de Programas em *Load Time*

Problemas

- A interface de invocação do programa é complexa.
- É difícil fazer a *memoization* de vários métodos em simultâneo.

Modificação de Programas em *Load Time*

Problemas

- A interface de invocação do programa é complexa.
- É difícil fazer a *memoization* de vários métodos em simultâneo.

Solução

```
public class Fib {

    @Memoized
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }

    ...
}
```

Anotações

Definição

- Informação acerca de um programa.
- Não são parte do programa.
- Não modificam a semântica do programa.
- Permitem anotar *packages*, classes, metodos, *fields*, parâmetros e variáveis.
- Podem ter parâmetros.
- Contrariamente ao Javadoc, as anotações podem ser preservadas pela compilação.
- Podem ser processadas em *compile time*, *load time* ou *run time*.
- A definição de uma anotação especifica (via meta-marcadores) a sua aplicabilidade (o *target*) e política de retenção (*retention*)

Três tipos de Anotações - Múltiplos Elementos

Definição da Anotação

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Foo {
    String bar();
    long baz();
}
```

Uso da Anotação

```
public class C1 {


    @Foo(bar = "Hello World", baz = 100)
    public void m1(int a, long b) {
        …
    }

}
```

Três tipos de Anotações - Único Elemento

Definição da Anotação

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Foo {
    String bar();
}
```

Uso da Anotação

```
public class C1 {


    @Foo("Hello World")
    public void m1(int a, long b) {
        …
    }

}
```

Três tipos de Anotações - Marcador

Definição da Anotação

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Foo {
}
```

Uso da Anotação

```
public class C1 {


    @Foo
    public void m1(int a, long b) {
        ...
    }

}
```

Anotações

Sintaxe

- Interfaces precedidas de `@`.
- Métodos com lista de parâmetros vazia e sem excepções.
- Tipo de retorno inclui apenas tipos primitivos, `String`, `Class`, *enums* e *arrays* destes tipos.

Anotações

Sintaxe

- Interfaces precedidas de `@`.
- Métodos com lista de parâmetros vazia e sem excepções.
- Tipo de retorno inclui apenas tipos primitivos, `String`, `Class`, *enums* e *arrays* destes tipos.

Anotações Pré-definidas - Simples

- `@Override`
- `@Deprecated`
- `@Suppresswarnings({` $warning_0$, ..., $warning_n$ `})`

Anotações

Anotações Pré-definidas - Meta-Anotações - @Target

- ElementType.TYPE
- ElementType.FIELD
- ElementType.METHOD
- ElementType.PARAMETER
- ElementType.CONSTRUCTOR
- ElementType.LOCAL_VARIABLE
- ElementType.ANNOTATION_TYPE

Anotações Pré-definidas - Meta-Anotações - @Retention

- RetentionPolicy.SOURCE
- RetentionPolicy.CLASS
- RetentionPolicy.RUNTIME

Modificação de Programas Anotados

Versão Anterior

```
import javassist.*;
import java.io.*;
import java.lang.reflect.*;

public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
            Class rtClass = ctClass.toClass();
            Method main =
                rtClass.getMethod("main",
                                  new Class[] { args.getClass() });
            String[] restArgs = new String[args.length - 2];
            System.arraycopy(args, 2, restArgs, 0, restArgs.length);
            main.invoke(null, new Object[] { restArgs });
        }
    }
}
```

Modificação de Programas Anotados

Versão Actual - Métodos Anotados

```
import javassist.*;
import java.io.*;
import java.lang.reflect.*;

public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoizeMethods(ctClass);
            Class rtClass = ctClass.toClass();
            Method main =
                rtClass.getMethod("main",
                                  new Class[] { args.getClass() });
            String[] restArgs = new String[args.length - 1];
            System.arraycopy(args, 1, restArgs, 0, restArgs.length);
            main.invoke(null, new Object[] { restArgs });
        }
    }
}
```

Modificação de Programas Anotados

Versão Actual - Métodos Anotados

```
static void memoizeMethods(CtClass ctClass) throws … {
    for (CtMethod ctMethod : ctClass.getDeclaredMethods()) {
        Object[] annotations = ctMethod.getAnnotations();
        if ((annotations.length == 1) &&
            (annotations[0] instanceof Memoized)) {
            memoize(ctClass, ctMethod);
        }
    }
}
```

Modificação de Programas Anotados

Versão Actual - Métodos Anotados

```
static void memoize(CtClass ctClass, CtMethod ctMethod)
    throws NotFoundException, CannotCompileException {
    String name = ctMethod.getName();
    CtField ctField =
        CtField.make("static java.util.Hashtable " +
                     name + "Results = " +
                     "    new java.util.Hashtable();",
                     ctClass);
    ctClass.addField(ctField);
    ctMethod.setName(name + "$original");
    ctMethod = CtNewMethod.copy(ctMethod, name, ctClass, null);
    ctMethod.setBody("{" +
                     "  Object result = " +
                     name + "Results.get($1);" +
                     "  if (result == null) {" +
                     "    result = " + name + "$original($$);" +
                     "    " + name + "Results.put($1, result);" +
                     "  }" +
                     "  return ($r)result;" +
                     "}");
    ctClass.addMethod(ctMethod);
}
```

Modificação de Programas Anotados

Problemas

- Apenas se pode ter uma classe com métodos anotados.
- É incómodo estar a especificar a classe cujos métodos estão anotados.
- O memorizador não pode actuar se a classe não for logo carregada.

Modificação de Programas Anotados

Problemas

- Apenas se pode ter uma classe com métodos anotados.
- É incómodo estar a especificar a classe cujos métodos estão anotados.
- O memorizador não pode actuar se a classe não for logo carregada.

Solução

Particularizar o *Class Loader*.

Carregamento de Tipos

Exemplo (Falta o nome da classe a executar)

```
$ java -verbose:class
```

Carregamento de Tipos

Exemplo (Falta o nome da classe a executar)

```
$ java -verbose:class
[Opened /usr/lib/jvm/java-1.5.0-sun-1.5.0.13/jre/lib/rt.jar]
[Opened /usr/lib/jvm/java-1.5.0-sun-1.5.0.13/jre/lib/jsse.jar]
[Opened /usr/lib/jvm/java-1.5.0-sun-1.5.0.13/jre/lib/jce.jar]
[Opened /usr/lib/jvm/java-1.5.0-sun-1.5.0.13/jre/lib/charsets.jar]
[Loaded java.lang.Object from shared objects file]
[Loaded java.io.Serializable from shared objects file]
[Loaded java.lang.Comparable from shared objects file]
[Loaded java.lang.CharSequence from shared objects file]
[Loaded java.lang.String from shared objects file]
[Loaded java.lang.reflect.GenericDeclaration from shared objects file]
[Loaded java.lang.reflect.Type from shared objects file]
[Loaded java.lang.reflect.AnnotatedElement from shared objects file]
[Loaded java.lang.Class from shared objects file]
[Loaded java.lang.Cloneable from shared objects file]
[Loaded java.lang.ClassLoader from shared objects file]
[Loaded java.lang.System from shared objects file]
[Loaded java.lang.Throwable from shared objects file]
[Loaded java.lang.Error from shared objects file]
[Loaded java.lang.ThreadDeath from shared objects file]
[Loaded java.lang.Exception from shared objects file]
[Loaded java.lang.RuntimeException from shared objects file]
[Loaded java.security.ProtectionDomain from shared objects file]
```

Carregamento

Exemplo

```
[Loaded java.lang.ClassNotFoundException from shared objects file]
[Loaded java.lang.LinkageError from shared objects file]
[Loaded java.lang.NoClassDefFoundError from shared objects file]
[Loaded java.lang.ClassCastException from shared objects file]
[Loaded java.lang.ArrayStoreException from shared objects file]
[Loaded java.lang.VirtualMachineError from shared objects file]
[Loaded java.lang.OutOfMemoryError from shared objects file]
[Loaded java.lang.StackOverflowError from shared objects file]
[Loaded java.lang.ref.Reference from shared objects file]
[Loaded java.lang.ref.SoftReference from shared objects file]
[Loaded java.lang.ref.WeakReference from shared objects file]
[Loaded java.lang.ref.FinalReference from shared objects file]
[Loaded java.lang.ref.PhantomReference from shared objects file]
[Loaded java.lang.ref.Finalizer from shared objects file]
[Loaded java.lang.Runnable from shared objects file]
[Loaded java.lang.Thread from shared objects file]
[Loaded java.lang.ThreadGroup from shared objects file]
[Loaded java.util.Dictionary from shared objects file]
[Loaded java.util.Map from shared objects file]
[Loaded java.util.Hashtable from shared objects file]
[Loaded java.util.Properties from shared objects file]
[Loaded java.lang.reflect.AccessibleObject from shared objects file]
[Loaded java.lang.reflect.Member from shared objects file]
```

Carregamento

Exemplo

```
[Loaded java.lang.reflect.Field from shared objects file]
[Loaded java.lang.reflect.Method from shared objects file]
[Loaded java.lang.reflect.Constructor from shared objects file]
[Loaded sun.reflect.MagicAccessorImpl from shared objects file]
[Loaded sun.reflect.MethodAccessor from shared objects file]
[Loaded sun.reflect.MethodAccessorImpl from shared objects file]
[Loaded sun.reflect.ConstructorAccessor from shared objects file]
[Loaded sun.reflect.ConstructorAccessorImpl from shared objects file]
[Loaded sun.reflect.DelegatingClassLoader from shared objects file]
[Loaded sun.reflect.ConstantPool from shared objects file]
[Loaded java.lang.Iterable from shared objects file]
[Loaded java.util.Collection from shared objects file]
[Loaded java.util.AbstractCollection from shared objects file]
[Loaded java.util.List from shared objects file]
[Loaded java.util.AbstractList from shared objects file]
[Loaded java.util.RandomAccess from shared objects file]
[Loaded java.util.Vector from shared objects file]
[Loaded java.lang.Appendable from shared objects file]
[Loaded java.lang.AbstractStringBuilder from shared objects file]
[Loaded java.lang.StringBuffer from shared objects file]
[Loaded java.lang.StackTraceElement from shared objects file]
[Loaded java.nio.Buffer from shared objects file]
[Loaded sun.misc.AtomicLong from shared objects file]
```

Carregamento

Exemplo

```
[Loaded java.lang.Boolean from shared objects file]
[Loaded java.lang.Character from shared objects file]
[Loaded java.lang.Number from shared objects file]
[Loaded java.lang.Float from shared objects file]
[Loaded java.lang.Double from shared objects file]
[Loaded java.lang.Byte from shared objects file]
[Loaded java.lang.Short from shared objects file]
[Loaded java.lang.Integer from shared objects file]
[Loaded java.lang.Long from shared objects file]
[Loaded java.lang.management.MemoryUsage from shared objects file]
[Loaded java.lang.StrictMath from shared objects file]
[Loaded java.io.ObjectStreamField from shared objects file]
[Loaded java.util.Comparator from shared objects file]
[Loaded java.security.Guard from shared objects file]
[Loaded java.security.Permission from shared objects file]
[Loaded java.security.BasicPermission from shared objects file]
[Loaded java.lang.RuntimePermission from shared objects file]
[Loaded java.util.AbstractMap from shared objects file]
[Loaded sun.misc.SoftCache from shared objects file]
[Loaded java.lang.ref.ReferenceQueue from shared objects file]
[Loaded java.lang.ref.ReferenceQueue$Null from shared objects file]
[Loaded java.lang.ref.ReferenceQueue$Lock from shared objects file]
[Loaded java.util.HashMap from shared objects file]
```

Carregamento

Exemplo

```
[Loaded java.io.ObjectStreamClass from shared objects file]
[Loaded java.security.PrivilegedAction from shared objects file]
[Loaded java.security.AccessController from shared objects file]
[Loaded java.util.Stack from shared objects file]
[Loaded sun.reflect.ReflectionFactory from shared objects file]
[Loaded java.lang.IncompatibleClassChangeError from shared objects file]
[Loaded java.lang.NoSuchMethodError from shared objects file]
[Loaded java.lang.annotation.Annotation from shared objects file]
[Loaded java.util.Map$Entry from shared objects file]
[Loaded java.util.HashMap$Entry from shared objects file]
[Loaded java.lang.reflect.ReflectPermission from shared objects file]
[Loaded java.lang.ref.Reference$Lock from shared objects file]
[Loaded java.lang.ref.Reference$ReferenceHandler from shared objects file]
[Loaded java.lang.ref.Finalizer$FinalizerThread from shared objects file]
[Loaded java.util.Enumeration from shared objects file]
[Loaded java.util.Hashtable$EmptyEnumerator from shared objects file]
[Loaded java.util.Iterator from shared objects file]
[Loaded java.util.Hashtable$EmptyIterator from shared objects file]
[Loaded java.util.Hashtable$Entry from shared objects file]
[Loaded java.nio.charset.Charset from shared objects file]
[Loaded sun.nio.cs.FastCharsetProvider from shared objects file]
[Loaded sun.nio.cs.StandardCharsets from shared objects file]
[Loaded sun.util.PreHashedMap from shared objects file]
```

Carregamento

Exemplo

```
[Loaded sun.nio.cs.StandardCharsets$Aliases from shared objects file]
[Loaded sun.nio.cs.StandardCharsets$Classes from shared objects file]
[Loaded sun.nio.cs.StandardCharsets$Cache from shared objects file]
[Loaded java.lang.ThreadLocal from shared objects file]
[Loaded java.lang.StringBuilder from shared objects file]
[Loaded sun.nio.cs.HistoricallyNamedCharset from shared objects file]
[Loaded sun.nio.cs.UTF_8 from shared objects file]
[Loaded java.lang.reflect.Modifier from shared objects file]
[Loaded sun.reflect.LangReflectAccess from shared objects file]
[Loaded java.lang.reflect.ReflectAccess from shared objects file]
[Loaded java.lang.Class$1 from shared objects file]
[Loaded sun.reflect.Reflection from shared objects file]
[Loaded java.util.Collections from shared objects file]
[Loaded java.util.Random from shared objects file]
[Loaded java.util.concurrent.atomic.AtomicLong from shared objects file]
[Loaded sun.misc.Unsafe from shared objects file]
[Loaded java.util.Set from shared objects file]
[Loaded java.util.AbstractSet from shared objects file]
[Loaded java.util.Collections$EmptySet from shared objects file]
[Loaded java.util.Collections$EmptyList from shared objects file]
[Loaded java.util.Collections$EmptyMap from shared objects file]
[Loaded java.util.Collections$ReverseComparator from shared objects file]
[Loaded java.util.Collections$SynchronizedMap from shared objects file]
```

Carregamento

Exemplo

```
[Loaded sun.reflect.ReflectionFactory$1 from shared objects file]
[Loaded sun.reflect.NativeConstructorAccessorImpl from shared objects file]
[Loaded sun.misc.VM from shared objects file]
[Loaded java.lang.StringCoding from shared objects file]
[Loaded java.lang.ThreadLocal$ThreadLocalMap from shared objects file]
[Loaded java.lang.ThreadLocal$ThreadLocalMap$Entry from shared objects file]
[Loaded java.lang.StringCoding$StringDecoder from shared objects file]
[Loaded java.lang.StringCoding$CharsetSD from shared objects file]
[Loaded java.nio.charset.CharsetDecoder from shared objects file]
[Loaded sun.nio.cs.UTF_8$Decoder from shared objects file]
[Loaded java.nio.charset.CodingErrorAction from shared objects file]
[Loaded sun.nio.cs.Surrogate$Generator from shared objects file]
[Loaded sun.nio.cs.Surrogate from shared objects file]
[Loaded java.nio.charset.CoderResult from shared objects file]
[Loaded java.nio.charset.CoderResult$1 from shared objects file]
[Loaded java.nio.charset.CoderResult$2 from shared objects file]
[Loaded java.nio.ByteBuffer from shared objects file]
[Loaded java.nio.HeapByteBuffer from shared objects file]
[Loaded java.nio.Bits from shared objects file]
[Loaded java.nio.ByteOrder from shared objects file]
[Loaded java.lang.Readable from shared objects file]
[Loaded java.nio.CharBuffer from shared objects file]
[Loaded java.nio.HeapCharBuffer from shared objects file]
```

Carregamento

Exemplo

```
[Loaded sun.misc.Version from shared objects file]
[Loaded java.io.Closeable from shared objects file]
[Loaded java.io.InputStream from shared objects file]
[Loaded java.io.FileInputStream from shared objects file]
[Loaded java.io.FileDescriptor from shared objects file]
[Loaded java.io.Flushable from shared objects file]
[Loaded java.io.OutputStream from shared objects file]
[Loaded java.io.FileOutputStream from shared objects file]
[Loaded java.io.FilterInputStream from shared objects file]
[Loaded java.io.BufferedInputStream from shared objects file]
[Loaded java.io.FilterOutputStream from shared objects file]
[Loaded java.io.PrintStream from shared objects file]
[Loaded java.io.BufferedOutputStream from shared objects file]
[Loaded java.io.Writer from shared objects file]
[Loaded java.io.OutputStreamWriter from shared objects file]
[Loaded sun.nio.cs.StreamEncoder from shared objects file]
[Loaded sun.io.Converters from shared objects file]
[Loaded sun.security.action.GetPropertyAction from shared objects file]
[Loaded java.nio.charset.CharsetEncoder from shared objects file]
[Loaded sun.nio.cs.UTF_8$Encoder from shared objects file]
[Loaded sun.nio.cs.Surrogate$Parser from shared objects file]
[Loaded java.io.BufferedWriter from shared objects file]
[Loaded java.lang.Runtime from shared objects file]
```

Carregamento

Exemplo

```
[Loaded java.io.File from shared objects file]
[Loaded java.io.FileSystem from shared objects file]
[Loaded java.io.UnixFileSystem from shared objects file]
[Loaded java.io.ExpiringCache from shared objects file]
[Loaded java.util.LinkedHashMap from shared objects file]
[Loaded java.io.ExpiringCache$1 from shared objects file]
[Loaded java.util.LinkedHashMap$Entry from shared objects file]
[Loaded java.lang.ClassLoader$3 from shared objects file]
[Loaded java.lang.StringCoding$StringEncoder from shared objects file]
[Loaded java.lang.StringCoding$CharsetSE from shared objects file]
[Loaded java.io.ExpiringCache$Entry from shared objects file]
[Loaded java.lang.ClassLoader$NativeLibrary from shared objects file]
[Loaded java.lang.Terminator from shared objects file]
[Loaded sun.misc.SignalHandler from shared objects file]
[Loaded sun.misc.Signal from shared objects file]
[Loaded sun.misc.NativeSignalHandler from shared objects file]
[Loaded sun.misc.JavaLangAccess from shared objects file]
[Loaded java.lang.System$2 from shared objects file]
[Loaded sun.misc.SharedSecrets from shared objects file]
[Loaded java.lang.NullPointerException from shared objects file]
[Loaded java.lang.ArithmeticException from shared objects file]
[Loaded java.lang.Compiler from shared objects file]
[Loaded java.lang.Compiler$1 from shared objects file]
```

Carregamento

Exemplo

```
[Loaded sun.misc.Launcher from shared objects file]
[Loaded java.net.URLStreamHandlerFactory from shared objects file]
[Loaded sun.misc.Launcher$Factory from shared objects file]
[Loaded java.security.SecureClassLoader from shared objects file]
[Loaded java.net.URLClassLoader from shared objects file]
[Loaded sun.misc.Launcher$ExtClassLoader from shared objects file]
[Loaded sun.security.util.Debug from shared objects file]
[Loaded java.util.StringTokenizer from shared objects file]
[Loaded java.security.PrivilegedExceptionAction from shared objects file]
[Loaded sun.net.www.ParseUtil from shared objects file]
[Loaded java.util.BitSet from shared objects file]
[Loaded java.lang.Math from shared objects file]
[Loaded java.net.URL from shared objects file]
[Loaded java.util.Locale from shared objects file]
[Loaded java.lang.CharacterDataLatin1 from shared objects file]
[Loaded sun.misc.Launcher$AppClassLoader$1 from shared objects file]
[Loaded java.lang.SystemClassLoaderAction from shared objects file]
[Loaded java.lang.Shutdown from shared objects file]
[Loaded java.lang.Shutdown$Lock from shared objects file]
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
```

Ciclo de Vida dos Tipos em Java

Fases

1. *Loading*: Localizar a representação binária de um tipo e carregamento para a JVM.
2. *Linking*: Incorporação de um tipo na JVM para execução.
3. *Initialization*: Execução dos inicializadores de um tipo (inicializadores estáticos das classes, inicializadores dos *fields* estáticos das classes e interfaces.
4. *Unloading*: Se um tipo é *unreachable* (não há quaisquer referências para ele) é elegível pela *garbage collection*.

Loading

Fases

1. Produção de uma *stream* binária que representa o tipo.
2. *Parsing* da *stream* para produzir estruturas internas guardadas na JVM e contendo toda a informação sobre os tipos carregados.
3. Criação da instância de `java.lang.Class` que representa o tipo.

Loading Time

- Não está especificado.
- Mas ocorre antes do *Linking Time* que ocorre antes do *Initialization Time* que ocorre antes do uso.
- Tipicamente, é atrasado para o mais tarde possível.
- Mas pode ser adiantado para o mais cedo possível (desde que isso seja possível).

Loading

Class Loaders

- São responsáveis pelo carregamento de tipos e estão organizados numa hierarquia (simples) de delegação.
- Em cada nó está um *Class Loader* cujo *parent* é o *Class Loader* que o carregou.
- Cada *Class Loader* pode (deve) delegar o carregamento de um tipo no seu *parent*.
- Se o *parent Class Loader* não consegue carregar um tipo, o *Class Loader* que delegou pode fazê-lo.
- O *Class Loader* que carrega um tipo fica associado ao tipo como o *Defining Class Loader* desse tipo.
- Cada *Class Loader* que delegou o carregamento de um tipo fica associado ao tipo como *Initiating Class Loader* desse tipo.

Loading

Class Loaders a partir de Java 1.2

A partir do Java 1.2, por omissão, a hieraquia de *Class Loader*s inclui:

1. *Bootstrap Class Loader* (também conhecido por *Primordial Class Loader*) para as classes fundamentais `java.*`, `javax.*`, etc.
2. *Extension Class Loader* (`ExtClassLoader`) para as classes contidas nas directorias de extensão do *runtime* (`java.ext.dirs`).
3. *System Class Loader* (`AppClassLoader`) para as classes contidas no *classpath* (`java.class.path`).

Loading

java.lang.ClassLoader

```
protected synchronized Class loadClass (String name, boolean resolve)
    throws ClassNotFoundException{
    // First check if the class is already loaded
    Class c = findLoadedClass(name);
    if (c == null) {
        try {
            if (parent != null) {
                c = parent.loadClass(name, false);
            } else {
                c = findBootstrapClass0(name);
            }
        } catch (ClassNotFoundException e) {
            // If still not found, then invoke
            // findClass to find the class.
            c = findClass(name);
        }
    }
    if (resolve) {
        resolveClass(c); //linking
    }
    return c;
}
```

Loading

Identificação de um Tipo

- Cada tipo é identificado pelo seu *fully qualified name*.
- Cada tipo *carregado* é identificado pelo seu *fully qualified name* e *class loader*.
- Cada *class loader* constitui um *namespace* diferente.
- O mesmo tipo carregado por dois *class loader*s diferentes possui duas ocorrências.
- Cada ocorrência de uma classe associada a um *class loader* é incompatível com outras ocorrências da mesma classe associadas a outros *class loaders*.
- A incompatibilidade não tem a ver com o **tipo** de *class loader* mas sim com a **instância** de *class loader*.

Loading

Bug

```
MyClassLoader myClassLoader = new MyClassLoader();
Class boxClass = myClassLoader.loadClass("Box");
Object obj = boxClass.newInstance();
Box box = (Box)obj;
```

Explicação

- O código foi carregado por um primeiro *class loader*.
- Esse primeiro *class loader* carregou a classe Box pois existe uma variável desse tipo.
- A execução do código cria um segundo *class loader* e usa-o para carregar a classe Box.
- Este segundo carregamento da classe Box fica associado ao segundo *class loader*.
- O *typecast* falha pois as classes são consideradas diferentes.

Loading

O *Class Loader* do Javassist

```
import javassist.*;
import Foo;

public class Main {

  public static void main(String[] args) throws Throwable {
     ClassPool pool = ClassPool.getDefault();
     //Create Javassist class loader
     Loader classLoader = new Loader(pool);
     //Obtain the compile time class Foo
     CtClass ctFoo = pool.get("Foo");

     //Modify class Foo
     ...

     //Obtain the run time class Foo
     Class rtFoo = classLoader.loadClass("Foo");
     //Instantiate Foo
     Object foo = rtFoo.newInstance();
     ...
  }
}
```

Loading

Listeners

- É possível associar *listeners* ao *class loader* do Javassist.
- Os *listeners* são notificados:
    - Quando são adicionados ao *class loader* (método `start`).
    - Sempre que uma classe vai ser carregada (método `onLoad`).
- Os *listeners* implementam a interface `javassist.Translator`:

`javassist.Translator`

```
public interface Translator {

    public void start(ClassPool pool)
        throws NotFoundException, CannotCompileException;

    public void onLoad(ClassPool pool, String classname)
        throws NotFoundException, CannotCompileException;
}
```

Loading

O *Class Loader* do Javassist

```
public class MyTranslator implements Translator {

    void start(ClassPool pool)
        throws NotFoundException, CannotCompileException {
        // Do noting
    }

    void onLoad(ClassPool pool, String className)
        throws NotFoundException, CannotCompileException {
        // Obtain the compile time class
        CtClass ctClass = pool.get(className);

         // Modify the class
        ...

        // That's all.  The class will now be automatically
        // loaded from the modified byte code
    }
}
```

Modificação de Programas Anotados

Versão Anterior

```
public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoizeMethods(ctClass);
            Class rtClass = ctClass.toClass();
            Method main =
                rtClass.getMethod("main",
                                  new Class[] { args.getClass() });
            String[] restArgs = new String[args.length - 1];
            System.arraycopy(args, 1, restArgs, 0, restArgs.length);
            main.invoke(null, new Object[] { restArgs });
        }
    }
}
```

Modificação de Programas Anotados

Versão com *Class Loader* do Javassist

```
public class MemoizeAndRun {

    public static void main(String[] args) throws … {
        if (args.length < 1) {
            …
        } else {
            Translator translator = new MemoizeTranslator();
            ClassPool pool = ClassPool.getDefault();
            Loader classLoader = new Loader();
            classLoader.addTranslator(pool, translator);
            String[] restArgs = new String[args.length - 1];
            System.arraycopy(args, 1, restArgs, 0, restArgs.length);
            classLoader.run(args[0], restArgs);
        }
    }
}
```

Modificação de Programas Anotados

Versão com *Class Loader* do Javassist

```
class MemoizeTranslator implements Translator {

    public void start(ClassPool pool)
        throws NotFoundException, CannotCompileException {
    }

    public void onLoad(ClassPool pool, String className)
        throws NotFoundException, CannotCompileException {
        CtClass ctClass = pool.get(className);
        try {
            memoizeMethods(ctClass);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    …
}
```

Modificação de Programas Anotados

Versão com *Class Loader* do Javassist

```
class MemoizeTranslator implements Translator {

    ...

    void memoizeMethods(CtClass ctClass)
        throws NotFoundException, CannotCompileException,
               ClassNotFoundException {
        for (CtMethod ctMethod : ctClass.getDeclaredMethods()) {
            Object[] annotations = ctMethod.getAnnotations();
            if ((annotations.length == 1) &&
                (annotations[0] instanceof Memoized)) {
                memoize(ctClass, ctMethod);
            }
        }
    }

    ...
}
```

Programas com História

Problema

- Pretendemos ser capazes de fazer *undo* da execução de programas Java.
- Pretendemos ser capazes de criar *checkpoints* capazes de representar o estado da execução de um programa Java.
- Pretendemos ser capazes de fazer recuar a execução de um programa Java até um dado *checkpoint*.

Programas com História

Uma pessoa tem um nome, uma idade e um amigo

```
class Person {
    String name;
    int age;
    Person friend;

    public String toString() {
        return "[" + name + "," + age +
            ((friend == null) ? "" : " with friend " + friend) +
            "]";
    }
}
```

Sim, eu sei:

- Falta construtor.
- Falta *getters* e *setters*.
- Não interessam para o exemplo.

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed is name to 'Louis'
//and got a friend
p0.name = "Louis";
p0.age = 53;
p1.age = 55;
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed is name to 'Louis'
//and got a friend
p0.name = "Louis";
p0.age = 53;
p1.age = 55;
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//25 years later, John (hum, I mean 'Louis') died
p1.age = 70;
p1.friend = null;
println(p1);//[Paul,70]
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed is name to 'Louis'
//and got a friend
p0.name = "Louis";
p0.age = 53;
p1.age = 55;
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//25 years later, John (hum, I mean 'Louis') died
p1.age = 70;
p1.friend = null;
println(p1);//[Paul,70]
//Let's go back in time
History.restoreState(state1);
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed is name to 'Louis'
//and got a friend
p0.name = "Louis";
p0.age = 53;
p1.age = 55;
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//25 years later, John (hum, I mean 'Louis') died
p1.age = 70;
p1.friend = null;
println(p1);//[Paul,70]
//Let's go back in time
History.restoreState(state1);
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
//and again
History.restoreState(state0);
println(p1);//[Paul,23 with friend [John,21]]
```

Programas com História

Guardar o estado do programa

```
import java.util.Stack;
import java.lang.reflect.*;

public class History {

    static Stack<ObjectFieldValue> undoTrail =
        new Stack<ObjectFieldValue>();

    public static void storePrevious(Object object,
                                     String className,
                                     String fieldName,
                                     Object value) {
        undoTrail.push(new ObjectFieldValue(object,
                                            className,
                                            fieldName,
                                            value));
    }

    ...
}
```

Programas com História

Guardar o estado do programa

```
import java.util.Stack;
import java.lang.reflect.*;

public class History {

    ...

    public static int currentState() {
        return undoTrail.size();
    }

    public static void restoreState(int state) {
        //undo all actions until size == state
        while (undoTrail.size() != state) {
            undoTrail.pop().restore();
        }
    }
}
```

Programas com História

Guardar o estado do programa

```
class ObjectFieldValue {
    Object object;
    String className;
    String fieldName;
    Object value;

    ObjectFieldValue(Object object,
                     String className,
                     String fieldName,
                     Object value) {
        this.object = object;
        this.className = className;
        this.fieldName = fieldName;
        this.value = value;
    }

    ...
}
```

Programas com História

Guardar o estado do programa

```
class ObjectFieldValue {

    ...

    void restore() {
        try {
            Field field =
                Class.forName(className).
                getDeclaredField(fieldName);
            field.setAccessible(true);
            field.set(object, value);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        } catch (NoSuchFieldException e) {
            throw new RuntimeException(e);
        } catch (IllegalAccessException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Programas com História

Javassist

```
import javassist.*;
import javassist.expr.*;
import java.io.*;
import java.lang.reflect.*;

public class Undoable {

    public static void main(String[] args) throws … {
        if (args.length < 1) {
            …
        } else {
            Translator translator = new UndoableTranslator();
            ClassPool pool = ClassPool.getDefault();
            Loader classLoader = new Loader();
            classLoader.addTranslator(pool, translator);
            String[] restArgs = new String[args.length - 1];
            System.arraycopy(args, 1, restArgs, 0, restArgs.length);
            classLoader.run(args[0], restArgs);
        }
    }
}
```

Programas com História

Javassist

```
class UndoableTranslator implements Translator {

    public void start(ClassPool pool)
        throws NotFoundException, CannotCompileException {
    }

    public void onLoad(ClassPool pool, String className)
        throws NotFoundException, CannotCompileException {
        CtClass ctClass = pool.get(className);
        makeUndoable(ctClass);
    }

    void makeUndoable(CtClass ctClass) {
        ...
    }
}
```

Programas com História

Javassist

```
void makeUndoable(CtClass ctClass)
    throws NotFoundException, CannotCompileException {
    final String template =
        "{" +
        "  History.storePrevious($0, \"%s\",\"%s\", ($w)$0.%s);" +
        "  $0.%s = $1;" +
        "}";
    for (CtMethod ctMethod : ctClass.getDeclaredMethods()) {
        ctMethod.instrument(new ExprEditor() {
                public void edit(FieldAccess fa)
                    throws CannotCompileException {
                    if (fa.isWriter()) {
                        String name = fa.getFieldName();
                        fa.replace(String.format(template,
                                                 fa.getClassName(),
                                                 name, name, name));
                    }
                }
            });
    }
}
```

Robert J. Chassell.
An Introduction to Programming in Emacs Lisp.
GNU Press, pub-GNU-PRESS:adr, 2001.

S. Chiba.
Javassist -- A reflection-based programming wizard for java.
In *Proceedings of the Workshop on Reflective Programming in C++ at the 13th ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'98)*, Vancouver, Canada, October 1998.
http://www.csg.is.titech.ac.jp/ chiba/oopsla98/proc/chiba.pdf.

Shigeru Chiba.
Load-time structural reflection in Java.
Lecture Notes in Computer Science, 1850:313--??, 2000.

Sheng Liang and Gilad Bracha.
Dynamics class loading in the java virtual machine.
In *OOPSLA*, pages 36--44, 1998.

Pattie Maes.
Concepts and experiments in computational reflection.
In Norman Meyrowitz, editor, *Proceedings of the 2nd Annual Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '87)*, pages 147--155, Orlando, FL, USA, October 1987. ACM Press.